Developing a Data Harvester in the Amazon Cloud for the Automated Assimilation of Florida's Healthy Beaches Reports into the GCOOS Data Portal

Robert Currier, Barbara Kirkpatrick

Abstract— **The Florida Department of Health funds beach water sampling for 34 of Florida's coastal counties. Mote Marine Laboratory, in conjunction with the Gulf of Mexico Coastal Ocean Observing System (GCOOS) developed an automated data harvester that used web scraping technology to capture the reported data from each monitored county and store the data in a MySQL database. The database was queried nightly and the results were used to build XML files that were ingested by GCOOS and published through the GCOOS Data Portal. In early 2013 the Florida Department of Health outsourced their web site to a commercial service provider. The new Healthy Beaches web site used a mash-up of Google Maps and JavaScript code and no longer returned a standard HTML document. The conversion from traditional web page to dynamic mash-up rendered our data harvester inoperable. Our entire data pipeline had to be completely rebuilt.**

I. INTRODUCTION

The Florida Department of Health funds beach water sampling for 34 of Florida's coastal counties. Mote Marine Laboratory, in conjunction with the Gulf of Mexico Coastal Ocean Observing System (GCOOS) developed an automated data harvester that used web scraping technology to capture the reported data from each monitored county and store the data in a MySQL database. The database was queried nightly and the results were used to build XML files that were ingested by GCOOS and published through the GCOOS Data Portal. In early 2013 the Florida Department of Health outsourced their web site to a commercial service provider. The new Healthy Beaches web site [1] used a mash-up of Google Maps and JavaScript code and no longer returned a standard HTML document. The conversion from traditional web page to a dynamic mash-up rendered our data harvester inoperable. Our entire data pipeline had to be completely rebuilt.

It is important to note that, while this paper details the steps we took to regain access to the lost data, this is in no way a recommended method of systems design. The original design, as well as the updated system, was built with no input from the Florida Department of Health. Typically, the data provider and the data consumer(s) would negotiate an API (applications programming interface) for data access and retrieval. A published API would have eliminated the need for the majority of the steps we took to rebuild our data pipeline. What we intend to demonstrate with this paper is that, in the absence of a published API, it is still possible to build a robust data retrieval application, with the caveat that substantial risks are involved, of which the most detrimental is that the data provider can change the data formatting without warning.

This work was supported by the Gulf Coast Ocean Observing System.

R. Currier is with Mote Marine Laboratory, Sarasota, FL 34236 USA (phone: 941-388-4441; e-mail: rdc@mote.org).

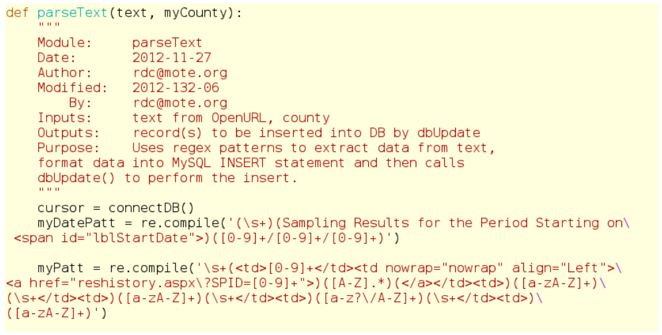
B. Kirkpatrick was with Mote Marine Laboratory, Sarasota, FL 34236 USA. She is now with the Gulf Coast Ocean Observing System, College Station TX 77843 USA (e-mail: barb.kirkpatrick@gcoos.org).

II. ORIGINAL SYSTEM DESCRIPTION

The original data harvester was written in the Python [2] programming language and used the urllib [3] library. Urllib provides a high-level interface for fetching data across the World Wide Web. MySQL [4] was used as the data storage engine. The harvester was run nightly out of cron, the Unix time-based job scheduler.

When called, the data harvester walked through a list of counties and used urllib to call the following link for each county: http://esetappsdoh.doh.state.fl.us/irm00beachwater/beachresults.apx?county='county.' Urlib retrieved the data generated by the link and returned the data as a Python text object.

The text object was fed to a regular expression that searched the text for matches based on the table style and formatting used by the Department Of Health site. The regex was complex and specific to this version of the Healthy Beaches web site.



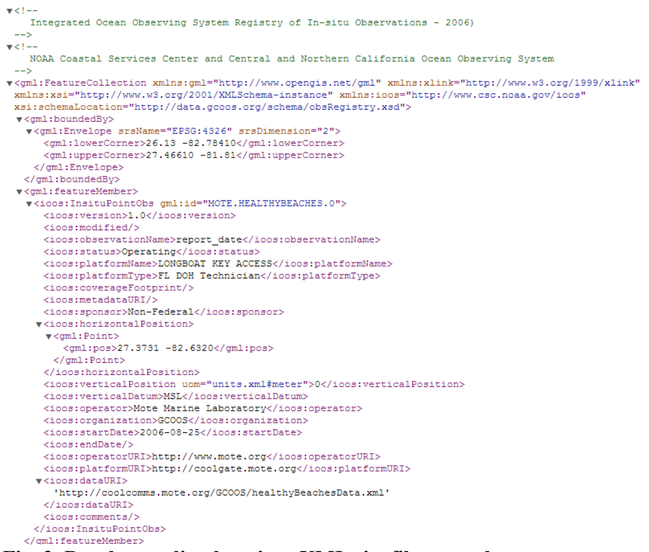
```
def parseText(text, myCounty):
    """
    Module:     parseText
    Date:       2012-11-27
    Author:     rdc@mote.org
    Modified:   2012-132-06
        By:     rdc@mote.org
    Inputs:     text from OpenURL, county
    Outputs:    record(s) to be inserted into DB by dbUpdate
    Purpose:    Uses regex patterns to extract data from text,
    format data into MySQL INSERT statement and then calls
    dbUpdate() to perform the insert.
    """
    cursor = connectDB()
    myDatePatt = re.compile('(\s+)(Sampling Results for the Period Starting on\
 <span id="lblStartDate">)([0-9]+/[0-9]+/[0-9]+)')

    myPatt = re.compile('\s+(<td>[0-9]+</td><td nowrap="nowrap" align="Left">\
<a href="reshistory.aspx\?SPID=[0-9]+">)([A-Z].*)(</a></td><td>)([a-zA-Z]+)\
(\s+</td><td>)([a-zA-Z]+)(\s+</td><td>)([a-z?\/A-Z]+)(\s+</td><td>)\
([a-zA-Z]+)')
```

Fig. 1. Example of regular expressions used to extract data

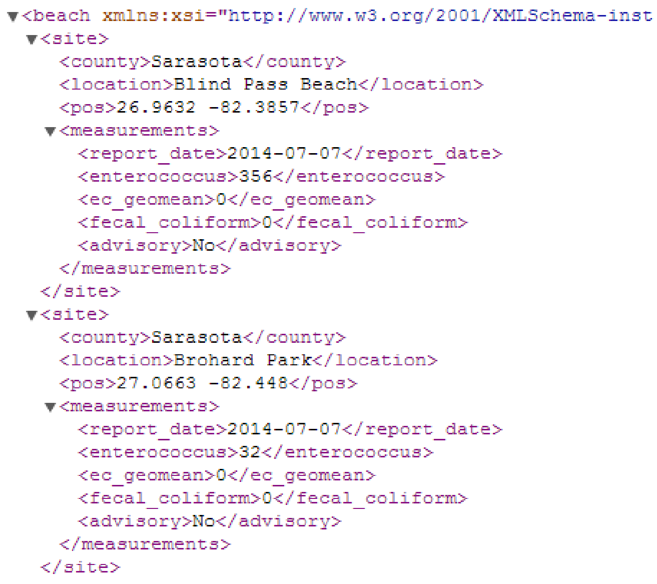
After the results for each beach were parsed from the text by the regular expression engine, the data was written to a MySQL database. Duplicate records were discarded.

Following a successful write to the database a routine was called that generated two IOOS-compliant XML files. The first file contained a list of all healthy beach sample locations. The second file contained the data for each of the sample locations. The GCOOS data harvester retrieved this data every day and ingested the data into the GCOOS Data Portal. [5]



```
▼<!--
   Integrated Ocean Observing System Registry of In-situ Observations - 2006)
 -->
▼<!--
   NOAA Coastal Services Center and Central and Northern California Ocean Observing System
 -->
▼<gml:FeatureCollection xmlns:gml="http://www.opengis.net/gml" xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ioos="http://www.csc.noaa.gov/ioos"
 xsi:schemaLocation="http://data.gcoos.org/schema/obsRegistry.xsd">
  ▼<gml:boundedBy>
    ▼<gml:Envelope srsName="EPSG:4326" srsDimension="2">
       <gml:lowerCorner>26.13 -82.78410</gml:lowerCorner>
       <gml:upperCorner>27.46610 -81.81</gml:upperCorner>
     </gml:Envelope>
   </gml:boundedBy>
  ▼<gml:featureMember>
    ▼<ioos:InsituPointObs gml:id="MOTE.HEALTHYBEACHES.0">
       <ioos:version>1.0</ioos:version>
       <ioos:modified/>
       <ioos:observationName>report_date</ioos:observationName>
       <ioos:status>Operating</ioos:status>
       <ioos:platformName>LONGBOAT KEY ACCESS</ioos:platformName>
       <ioos:platformType>FL DOH Technician</ioos:platformType>
       <ioos:coverageFootprint/>
       <ioos:metadataURI/>
       <ioos:sponsor>Non-Federal</ioos:sponsor>
      ▼<ioos:horizontalPosition>
        ▼<gml:Point>
           <gml:pos>27.3731 -82.6320</gml:pos>
         </gml:Point>
       </ioos:horizontalPosition>
       <ioos:verticalPosition uom="units.xml#meter">0</ioos:verticalPosition>
       <ioos:verticalDatum>MSL</ioos:verticalDatum>
       <ioos:operator>Mote Marine Laboratory</ioos:operator>
       <ioos:organization>GCOOS</ioos:organization>
       <ioos:startDate>2006-08-25</ioos:startDate>
       <ioos:endDate/>
       <ioos:operatorURI>http://www.mote.org</ioos:operatorURI>
       <ioos:platformURI>http://coolgate.mote.org</ioos:platformURI>
      ▼<ioos:dataURI>
         'http://coolcomms.mote.org/GCOOS/healthyBeachesData.xml'
       </ioos:dataURI>
       <ioos:comments/>
     </ioos:InsituPointObs>
   </gml:featureMember>
```

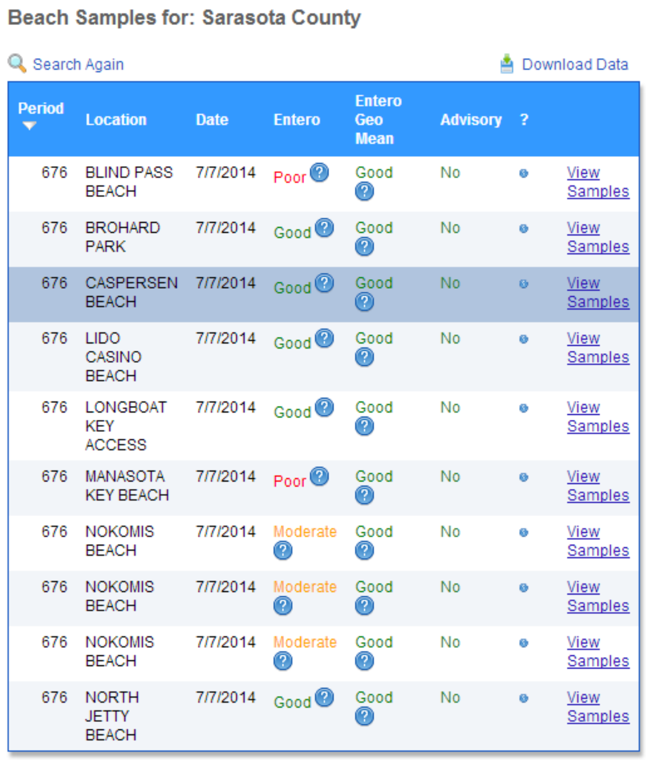
Fig. 2. Beach sampling locations XML site file example



```
▼<beach xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
  ▼<site>
     <county>Sarasota</county>
     <location>Blind Pass Beach</location>
     <pos>26.9632 -82.3857</pos>
    ▼<measurements>
       <report_date>2014-07-07</report_date>
       <enterococcus>356</enterococcus>
       <ec_geomean>0</ec_geomean>
       <fecal_coliform>0</fecal_coliform>
       <advisory>No</advisory>
     </measurements>
   </site>
  ▼<site>
     <county>Sarasota</county>
     <location>Brohard Park</location>
     <pos>27.0663 -82.448</pos>
    ▼<measurements>
       <report_date>2014-07-07</report_date>
       <enterococcus>32</enterococcus>
       <ec_geomean>0</ec_geomean>
       <fecal_coliform>0</fecal_coliform>
       <advisory>No</advisory>
     </measurements>
   </site>
```

Fig. 3. Beach sampling locations XML data file example

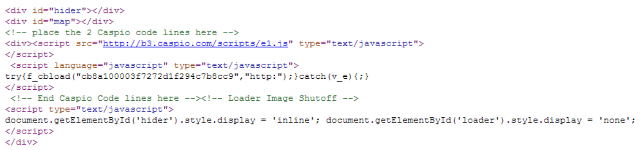
In early 2013 the Florida Department of Health outsourced their web site to a commercial service provider. The new Healthy Beaches web site used a mash-up of Google Maps and JavaScript code and no longer returned a standard HTML document when queried. The existing code depended on traditional HTML markup tags such as <table> and <div>, and required the data to be embedded in the HTML text. The new site returned no data but instead supplied proprietary JavaScript code that was executed in the browser. Urllib did not provide a browser-based environment and we were unable to execute the JavaScript and retrieve the data.



Beach Samples for: Sarasota County

Period	Location	Date	Entero	Entero Geo Mean	Advisory	?	
676	BLIND PASS BEACH	7/7/2014	Poor	Good	No	●	View Samples
676	BROHARD PARK	7/7/2014	Good	Good	No	●	View Samples
676	CASPERSEN BEACH	7/7/2014	Good	Good	No	●	View Samples
676	LIDO CASINO BEACH	7/7/2014	Good	Good	No	●	View Samples
676	LONGBOAT KEY ACCESS	7/7/2014	Good	Good	No	●	View Samples
676	MANASOTA KEY BEACH	7/7/2014	Poor	Good	No	●	View Samples
676	NOKOMIS BEACH	7/7/2014	Moderate	Good	No	●	View Samples
676	NOKOMIS BEACH	7/7/2014	Moderate	Good	No	●	View Samples
676	NOKOMIS BEACH	7/7/2014	Moderate	Good	No	●	View Samples
676	NORTH JETTY BEACH	7/7/2014	Good	Good	No	●	View Samples

Fig. 4. Browser-rendered data supplied by DOH web site



```
<div id="hider"></div>
<div id="map"></div>
<!-- place the 2 Caspio code lines here -->
<div><script src="http://b3.caspio.com/scripts/e1.js" type="text/javascript">
</script>
 <script language="javascript" type="text/javascript">
try{f_cbload("cb8a100003f7272d1f294c7b8cc9","http:");}catch(v_e){;}
</script>
 <!-- End Caspio Code lines here --><!-- Loader Image Shutoff -->
<script type="text/javascript">
document.getElementById('hider').style.display = 'inline'; document.getElementById('loader').style.display = 'none';
</script>
</div>
```

Fig. 5. JavaScript code returned by new web site

III. BUILDING THE NEW SYSTEM

We elected to use the Selenium [6] web driver toolkit for the Python programming language to gain access to the data generated by the JavaScript code. Selenium is a set of tools used to automate web browsers across many platforms and programming languages. The Selenium toolkit allowed us to programmatically control the FireFox browser and execute the JavaScript supplied by the new Department of Health web site. The executed JavaScript code generated a complex HTML document that was unsuitable for standard search and retrieval methods.

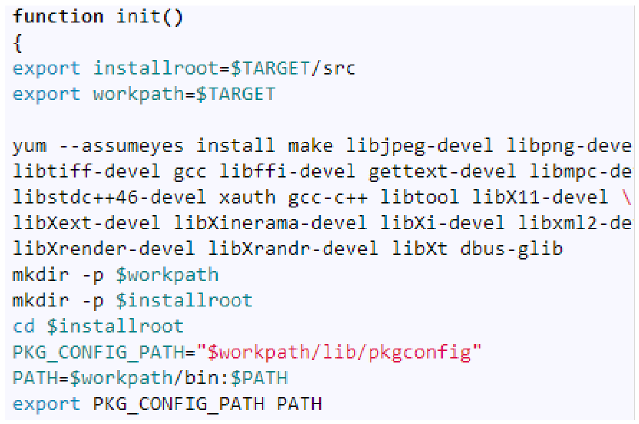
We fed this document to the BeautifulSoup4 [7] Python library and produced a standardized HTML document suitable for parsing. Using regular expressions [8], we matched on the data in the HTML document and extracted the relevant text for each water sampling site. Multiple passes through the HTML document were required as individual variables were not located on the same line of text. After all variables for a site were collected a Python object was created for each water sampling site. The objects were iterated over and the data records were stored in a MySQL database.

IV. MIGRATION TO THE AMAZON CLOUD

Our development environment for the new system was a Fedora Core 17 [9] virtual machine running under the VMware Player. We wrote the code and tested the system using this environment. The Selenium web driver worked as advertised and we prepared to deploy the code on our production server, an Amazon EC2 [10] cloud instance. It was during this step that we ran into a significant obstacle: Amazon EC2 instances are headless servers and lack graphics hardware and the software libraries required for rendering.

The browser we chose to use, Firefox, required the GTK+ toolkit. [11] GTK+, the GIMP toolkit, is a multi-platform toolkit for creating graphical interfaces. In order for Selenium to control a browser we first needed to install the GTK+ libraries and build Firefox from source code.

After several failed attempts to get GTK+ to build we found a very helpful script written by Joseph Lawson of http://joekiller.com. Run as the root user the script downloads and installs all the components needed for Selenium. The script also installs all the dependencies required by the Firefox browser. Using this script we were able to build GTK+ in two hours.



```
function init()
{
export installroot=$TARGET/src
export workpath=$TARGET

yum --assumeyes install make libjpeg-devel libpng-deve
libtiff-devel gcc libffi-devel gettext-devel libmpc-de
libstdc++46-devel xauth gcc-c++ libtool libX11-devel \
libXext-devel libXinerama-devel libXi-devel libxml2-de
libXrender-devel libXrandr-devel libXt dbus-glib
mkdir -p $workpath
mkdir -p $installroot
cd $installroot
PKG_CONFIG_PATH="$workpath/lib/pkgconfig"
PATH=$workpath/bin:$PATH
export PKG_CONFIG_PATH PATH
```

Fig. 6. Libraries required for GTK+ on Amazon EC2

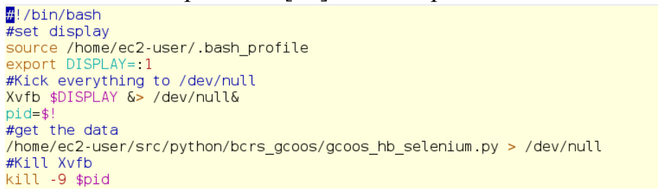
The script we used did an excellent job of downloading and building the GTK+ library, but it failed to build the FireFox browser. We were forced to download the source package for FireFox from the Mozilla repository on Github. [12] Firefox was built manually and installed in /usr/lib64. After adding /usr/lib64/firefox to our $PATH environment variable we had a working copy of Firefox on an Amazon EC2 instance.

With Selenium and Firefox working the final hurdle to deployment on the Amazon cloud was the lack of graphics hardware. While we had the libraries for display generation and the Firefox browser installed, we had nowhere to display the output. Amazon cloud instances are headless and have no display hardware.

The X virtual frame buffer, Xvfb [13] provided the solution. Xvfb is a display server that implements the X11 display protocol using in memory emulation. No graphics hardware is required. To a client application, Xvfb appears to be a normal display.

Installing and building Xvfb turned out to be the simplest part of this process. We were able to use the yum [14] package manager to download and install Xvfb by executing 'yum install xvfb.'

Xvfb operates by emulating a physical display in memory. In order for our application to use the Xvfb frame buffer it needed to be configured to point the frame buffer and not attempt to use the non-existent physical display. Once a virtual display is defined it remains in existence until it is destroyed, and further attempts to activate a display with the same id number will result in failure. We solved this problem with a simple Bash [15] shell script.



```
#!/bin/bash
#set display
source /home/ec2-user/.bash_profile
export DISPLAY=:1
#Kick everything to /dev/null
Xvfb $DISPLAY &> /dev/null&
pid=$!
#get the data
/home/ec2-user/src/python/bcrs_gcoos/gcoos_hb_selenium.py > /dev/null
#Kill Xvfb
kill -9 $pid
```

Fig. 7. Script to initiate and destroy the frame buffer

Upon execution, the script first sources the bash profile of the user running the script, ensuring that the script has access to the Firefox binaries. The DISPLAY environment variable is set to 1 and exported. Xvfb is called using the exported DISPLAY value, with the output sent to /dev/null so that no textual output appears. The process id is extracted from the '$!' environment variable and is used to destroy the Xvfb buffer. The Python script is then executed, and using the Selenium toolkit the script connects to the designated URL, retrieves the data, executes the embedded JavaScript and feeds the output to the BeautifulSoup4 library. After the data records for each sampling site have been extracted and written to the MySQL database, the Xvfb frame buffer is destroyed using the 'kill -9 $pid' command. The execution of the script is controlled by cron [16], the Unix time-based job scheduler.

V. CONCLUSION

We successfully solved a difficult problem using the described techniques and tools, but we cannot recommend this methodology as standard practice. We did not have access to the Department Of Health programming staff and did not have the funding to execute a collaborative project to share this data. We needed to find the simplest and fastest way to get data from the Healthy Beaches web site into the GCOOS Data Portal. If we were planning a similar project we would not consider replicating this solution as it is too fragile, as demonstrated by the failure of the original application when the web site was changed. A formally defined API and a collaborative team effort between the data provider and the data consumers is the preferred method for sharing data between organizations.

Acknowledgment

The authors wish to thank Joseph Lawson for his extremely helpful script. Without this script we would not have been able to build the GTK+ toolkit and our project would have failed. We also wish to acknowledge all of the developers that made contributions to the Free and Open Source projects used in the construction of our application. We especially want to recognize Seth Vidal, our good friend and the author of YUM. Seth was a dedicated contributor to the Fedora Project and Open Source community but was tragically killed while bicycling on July 8th, 2013.

References

[1] http://www.floridahealth.gov/%5C/healthy-environments/beachwater-quality/index/
[2] http://www.python.org/
[3] https://docs.python.org/2/library/urllib.html
[4] http://www.mysql.com/
[5] http://data.gcoos.org/
[6] http://docs.seleniumhq.org/projects/webdriver/
[7] https://pypi.python.org/pypi/beautifulsoup4/4.3.2/
[8] https://docs.python.org/2/library/re.html
[9] http://fedoraproject.org
[10] http://aws.amazon.com/ec2/
[11] http://www.gtk.org
[12] https://github.com/
[13] http://www.x.org/archive/X11R7.7/doc/man/man1/Xvfb.1.xhtml
[14] https://fedoraproject.org/wiki/Yum
[15] http://www.gnu.org/software/bash/
[16] http://en.wikipedia.org/wiki/Cron